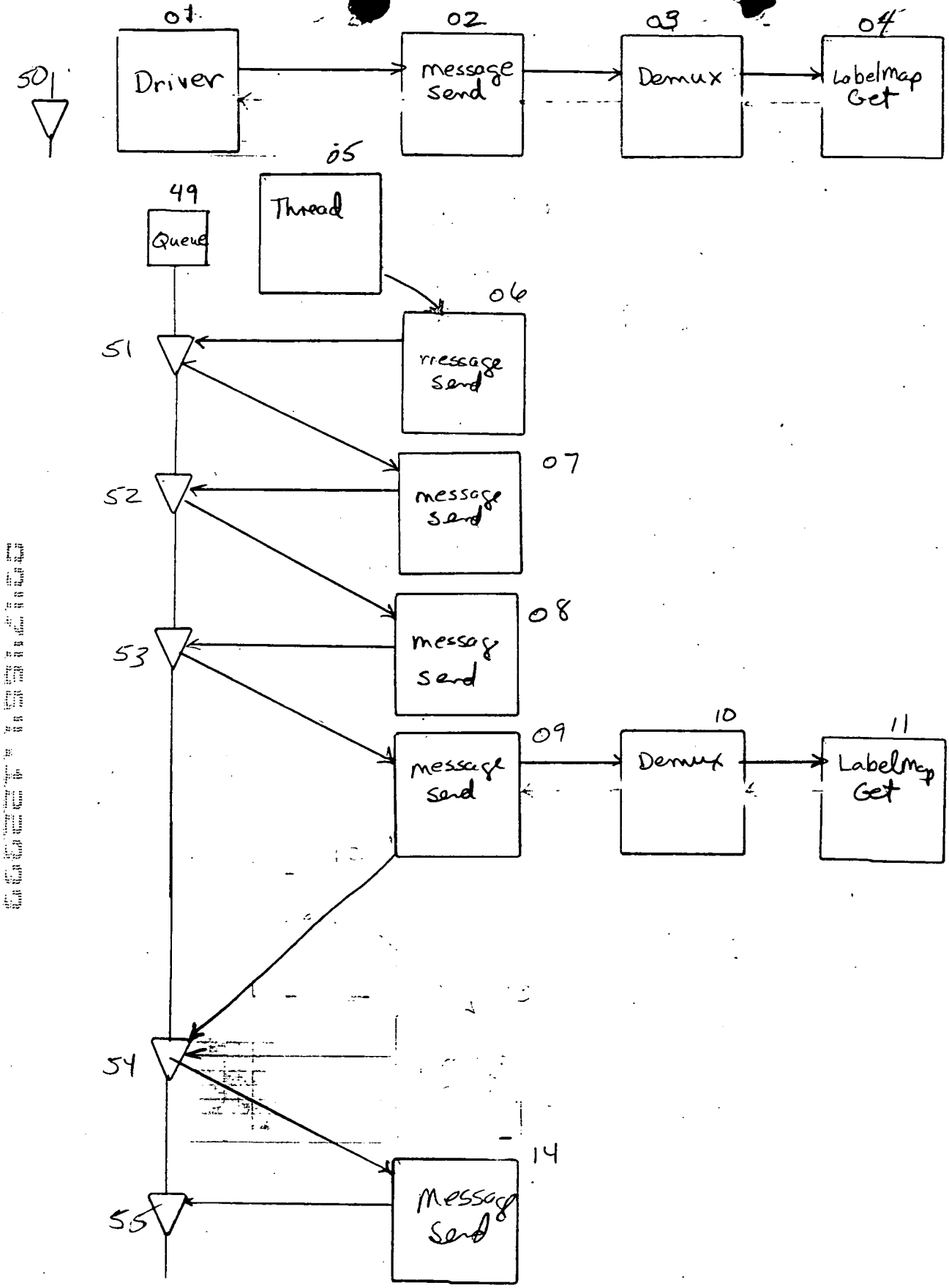

01
Driver
02
message send
03
Demux
04
LabelMap Get
50
05
Thread
49
Queue
06
message send
51
07
message send
52
08
message send
53
09
message send
10
Demux
11
LabelMap Get
54
14
message send
55

Fig 1

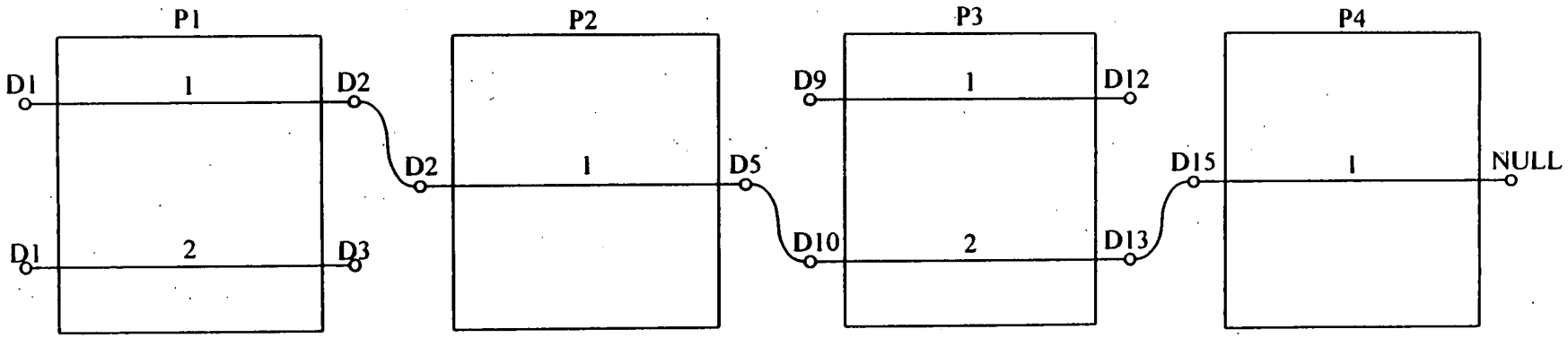


Fig. 2

SUBSTITUTE SHEET (RULE 26)

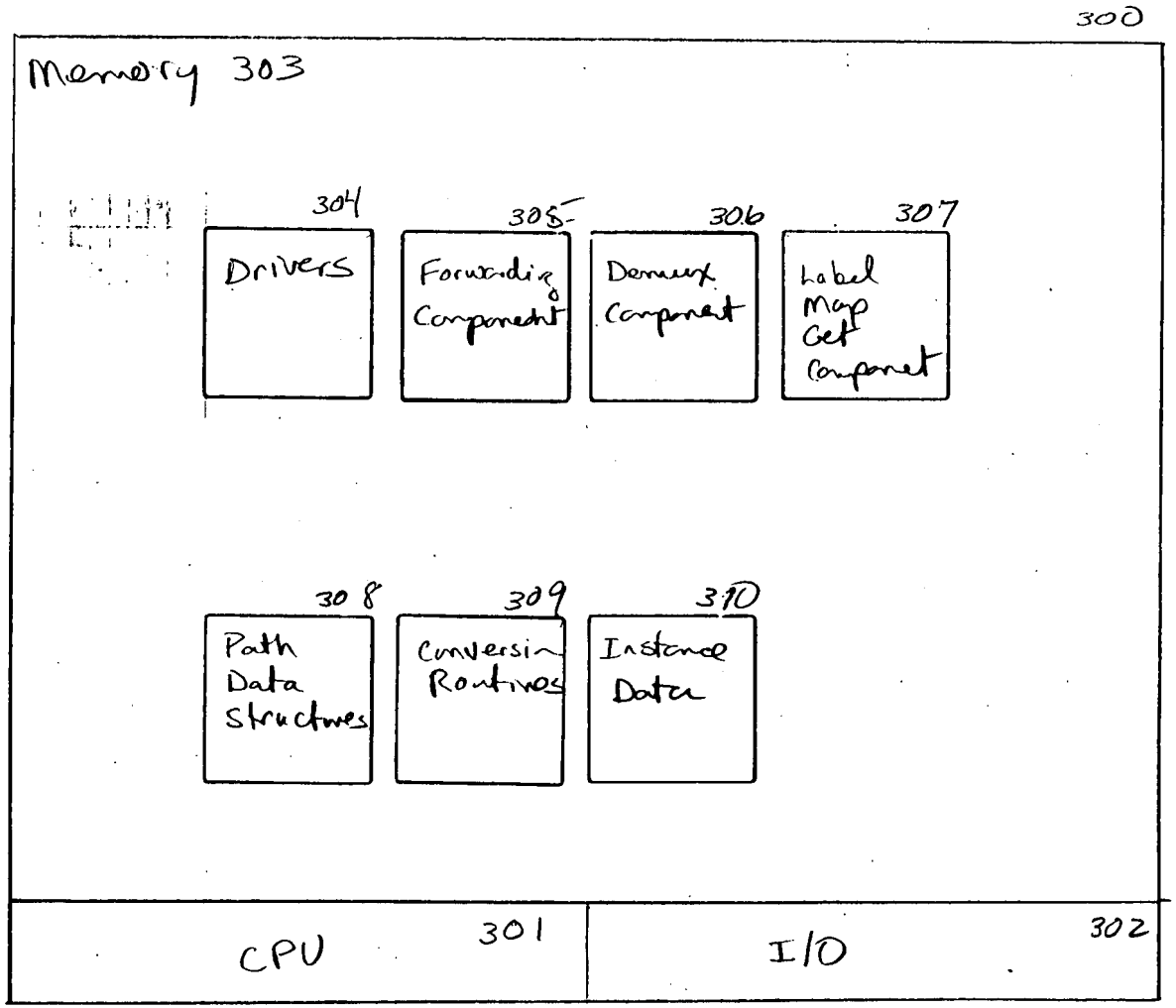


Figure 3

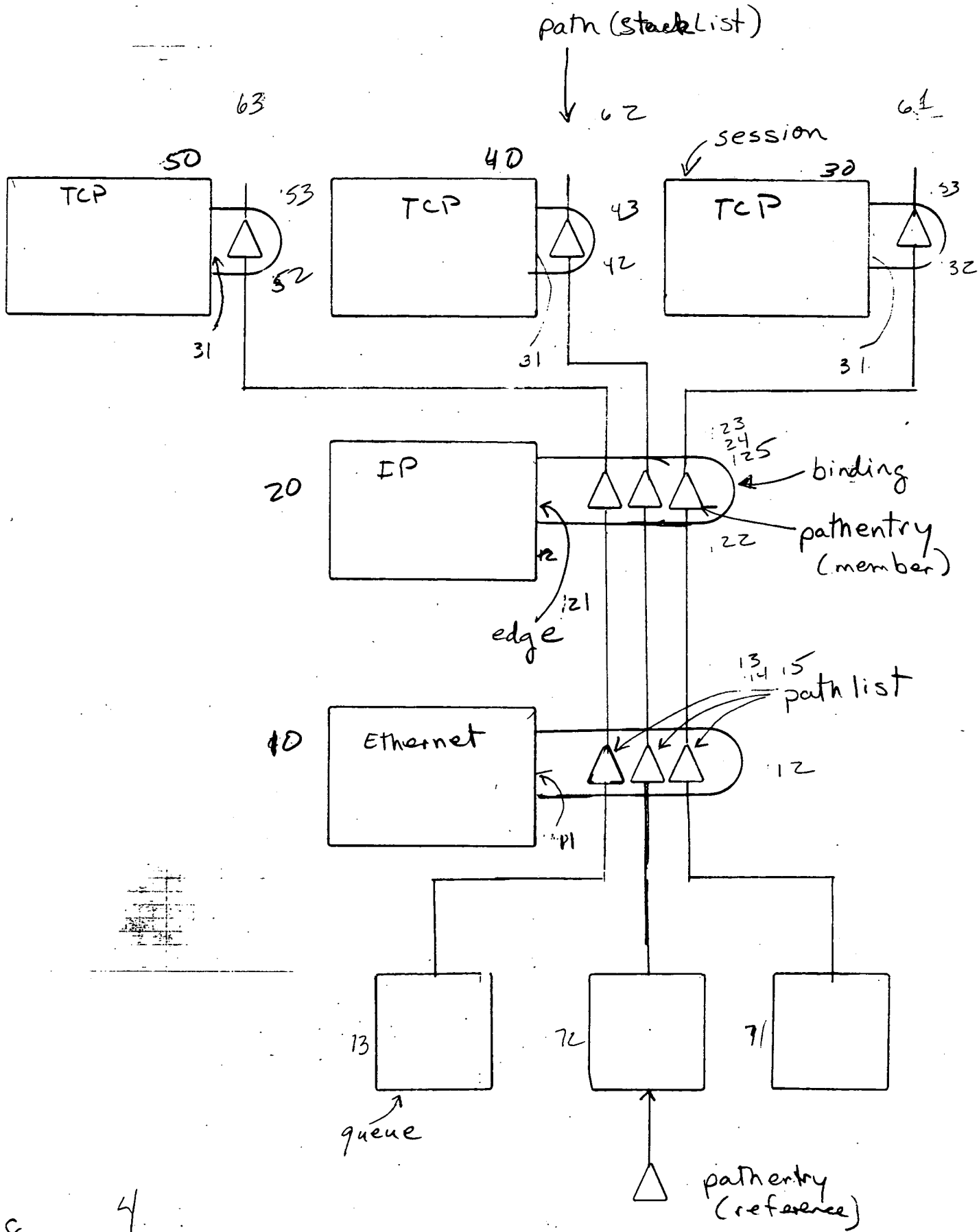

path (stackList)
63
62
61
session
50
40
30
TCP
TCP
TCP
53
52
43
42
33
32
31
31
31
23
24
25
binding
IP
20
pathentry
(member)
22
12
edge
21
13
14
15
path list
Ethernet
10
12
11
13
72
71
queue
pathentry
(reference)

Fig 4

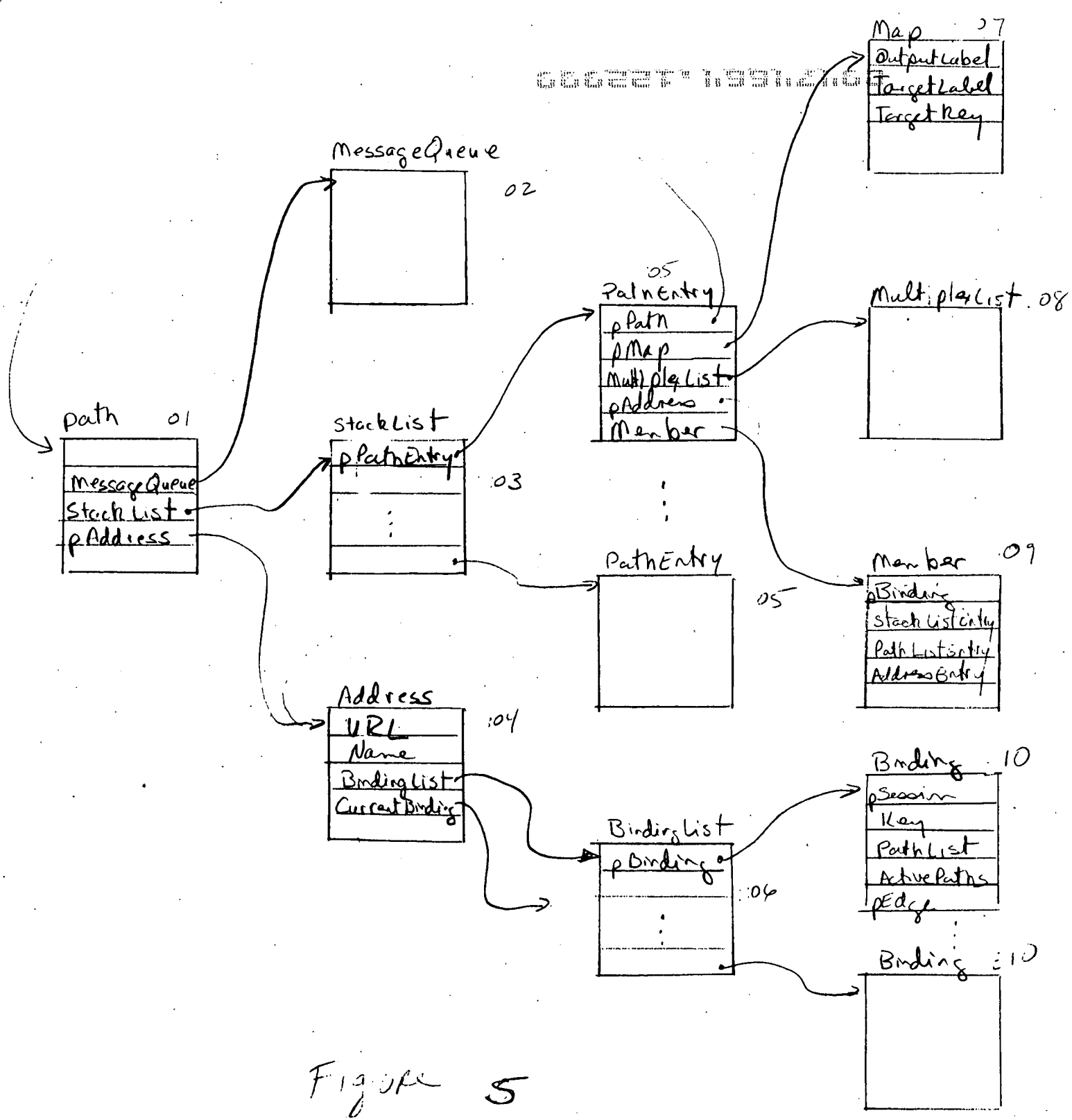

Map 07
OutputLabel
TargetLabel
TargetKey
MessageQueue
02
05
PathEntry
pPath
pMap
MultiplaqList
pAddress
Member
MultiplaqList 08
path 01
MessageQueue
StackList
pAddress
StackList
pPathEntry
03
PathEntry
05
Member 09
pBinding
StackListEntry
PathListEntry
AddressEntry
Address
URL
Name
BindingList
CurrentBinding
04
BindingList
pBinding
06
Binding 10
pSession
Key
PathList
ActivePaths
pEdge
Binding 10

Figure 5

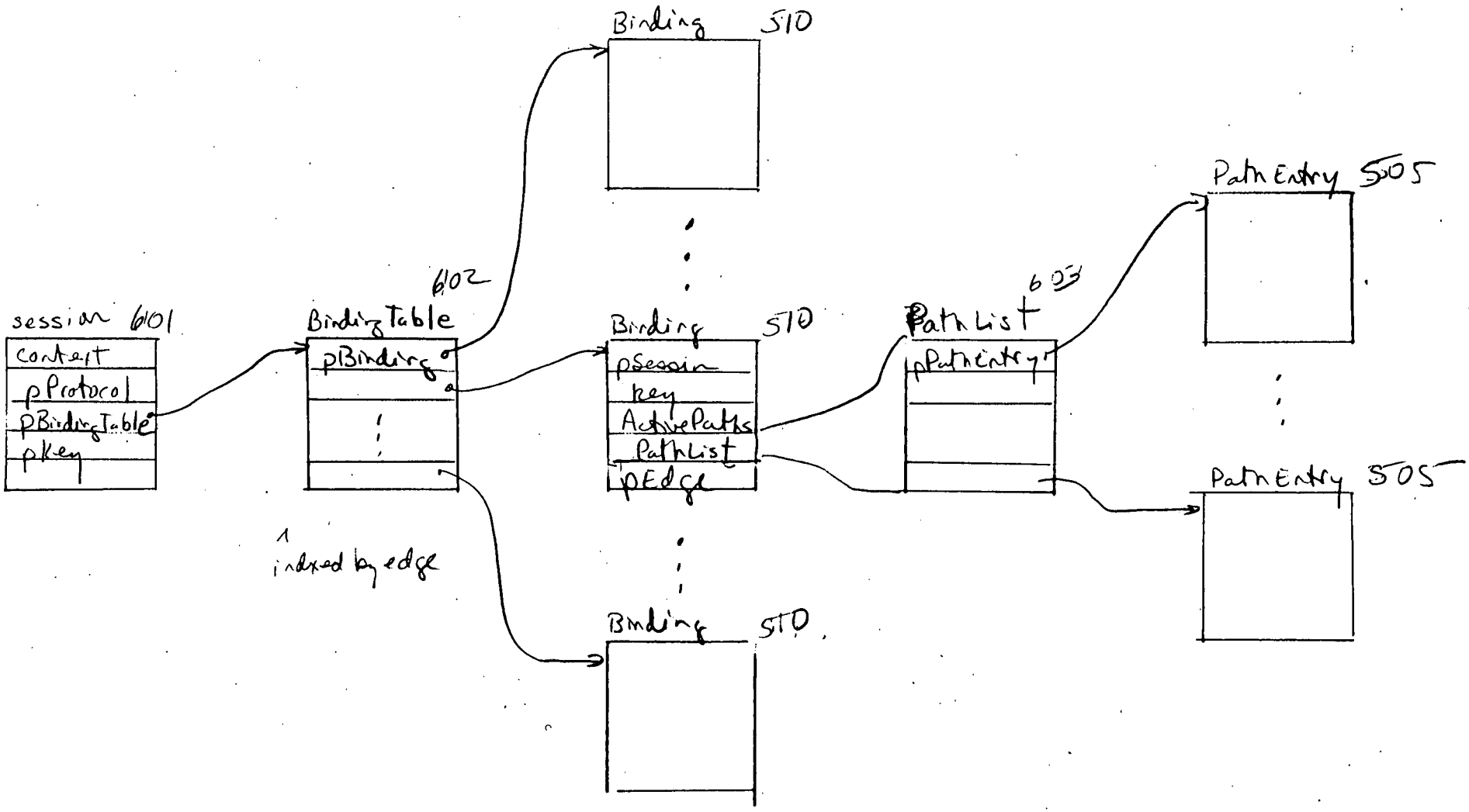

Binding 510
Path Entry 505
602
session 601
Binding Table
Binding 510
Path List
603
pBinding
context
pProtocol
pBindingTable
pKey
pSession
key
ActivePaths
PathList
pEdge
pPathEntry
Path Entry 505
indexed by edge
Binding 510

Figure 6

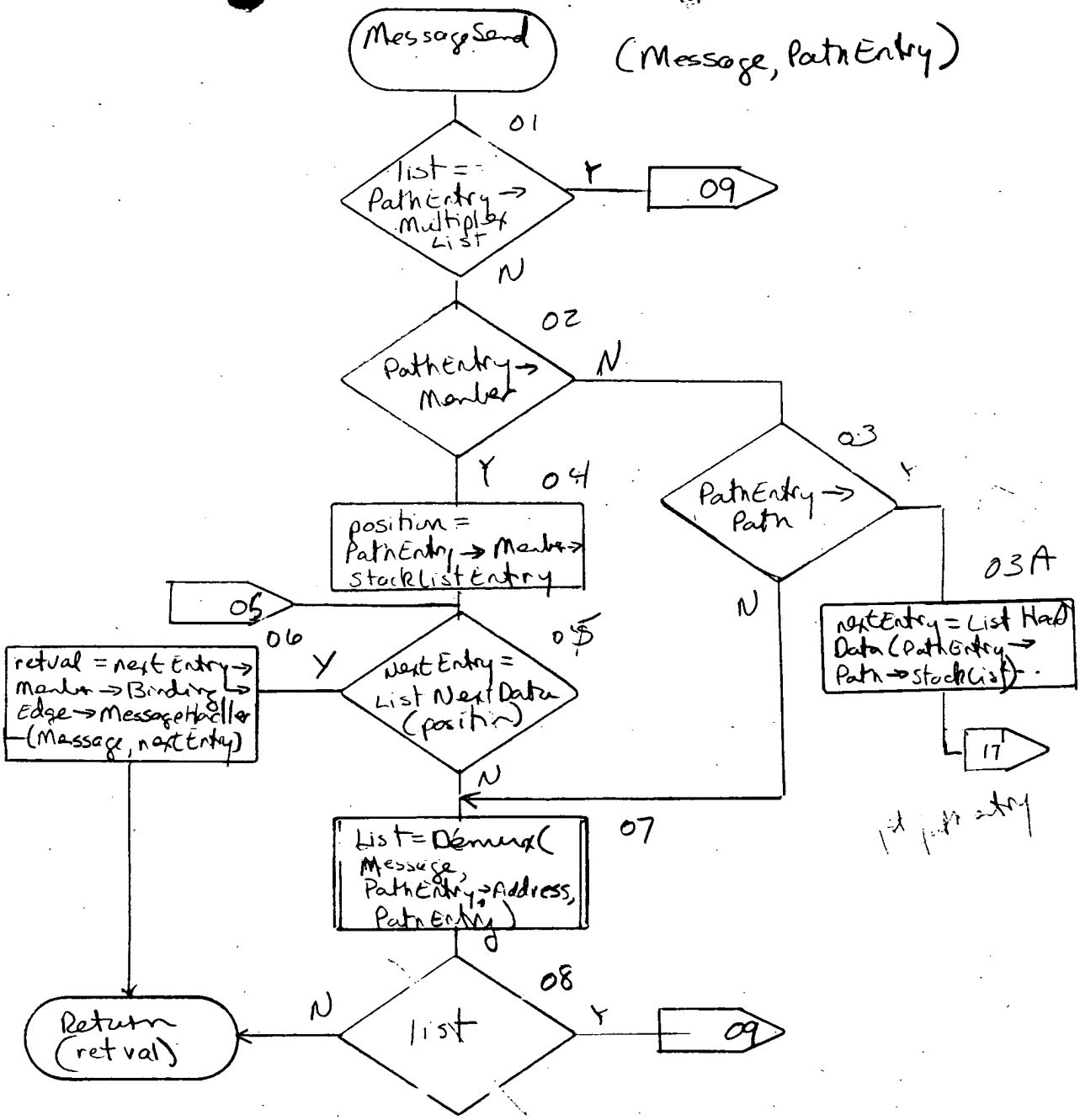

MessageSend
(Message, PathEntry)
01
list = PathEntry → multiplex List
Y
09
N
02
PathEntry → Marker
N
Y
04
position = PathEntry → Marker → StackListEntry
03
PathEntry → Path
Y
03A
nextEntry = List Head Data (PathEntry → Path → StackList)
17
N
05
05
06
Y
nextEntry = List Next Data (position)
retval = nextEntry → Marker → Binding → Edge → MessageHandler (Message, nextEntry)
N
07
List = Demux(Message, PathEntry → Address, PathEntry)
08
list
N
Y
09
Return (retval)

Figure 7A

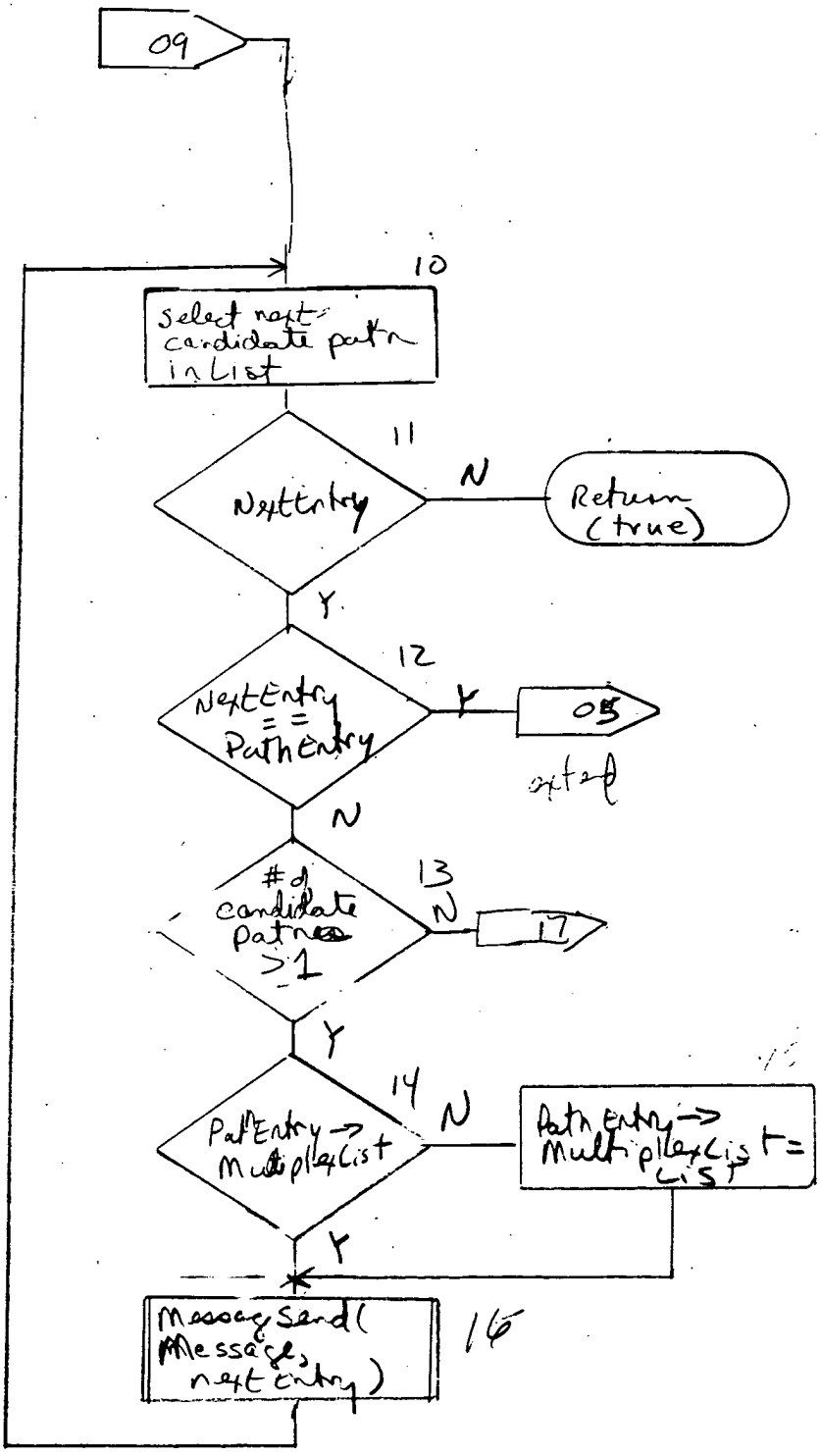
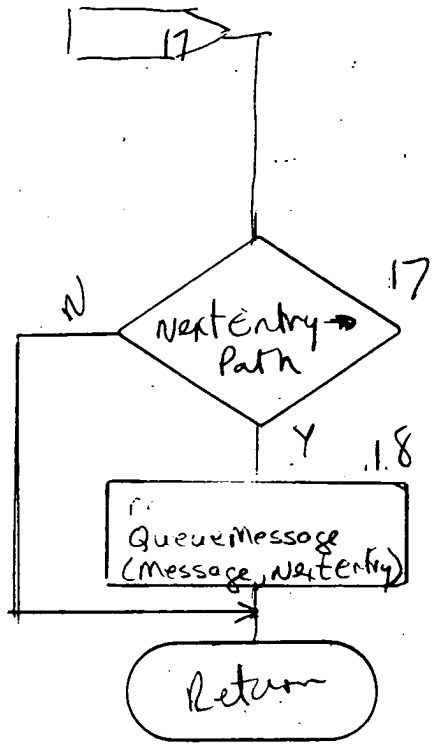

09
10
Select next candidate path in List
11
NextEntry
N
Return (true)
Y
12
NextEntry == PathEntry
Y
05
exit
N
13
# of candidate paths > 1
N
17
Y
14
PathEntry -> MultiplayList
N
PathEntry -> MultiplayList =
Y
MessageSend(Message, nextEntry)
16

Figure 7B



reference path entry

Figure 7C

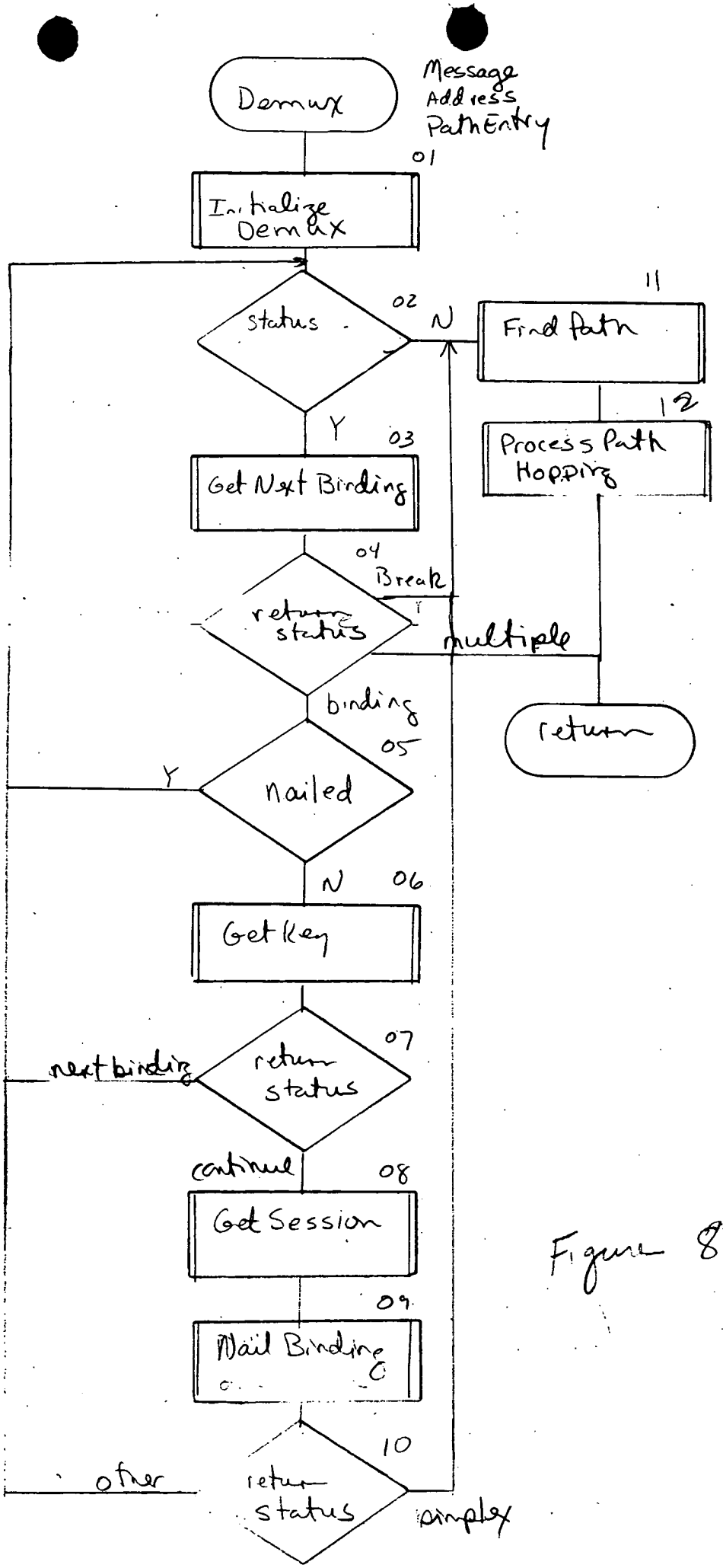

Demux
Message
Address
PathEntry
01
Initialize Demux
02
status
N
11
Find Path
12
Process Path Hopping
Y
03
Get Next Binding
04
Break
return status
multiple
return
binding
05
nailed
Y
N
06
Get Key
07
return status
next binding
continue
08
Get Session
09
Mail Binding
10
return status
other
simplex

Figure 8

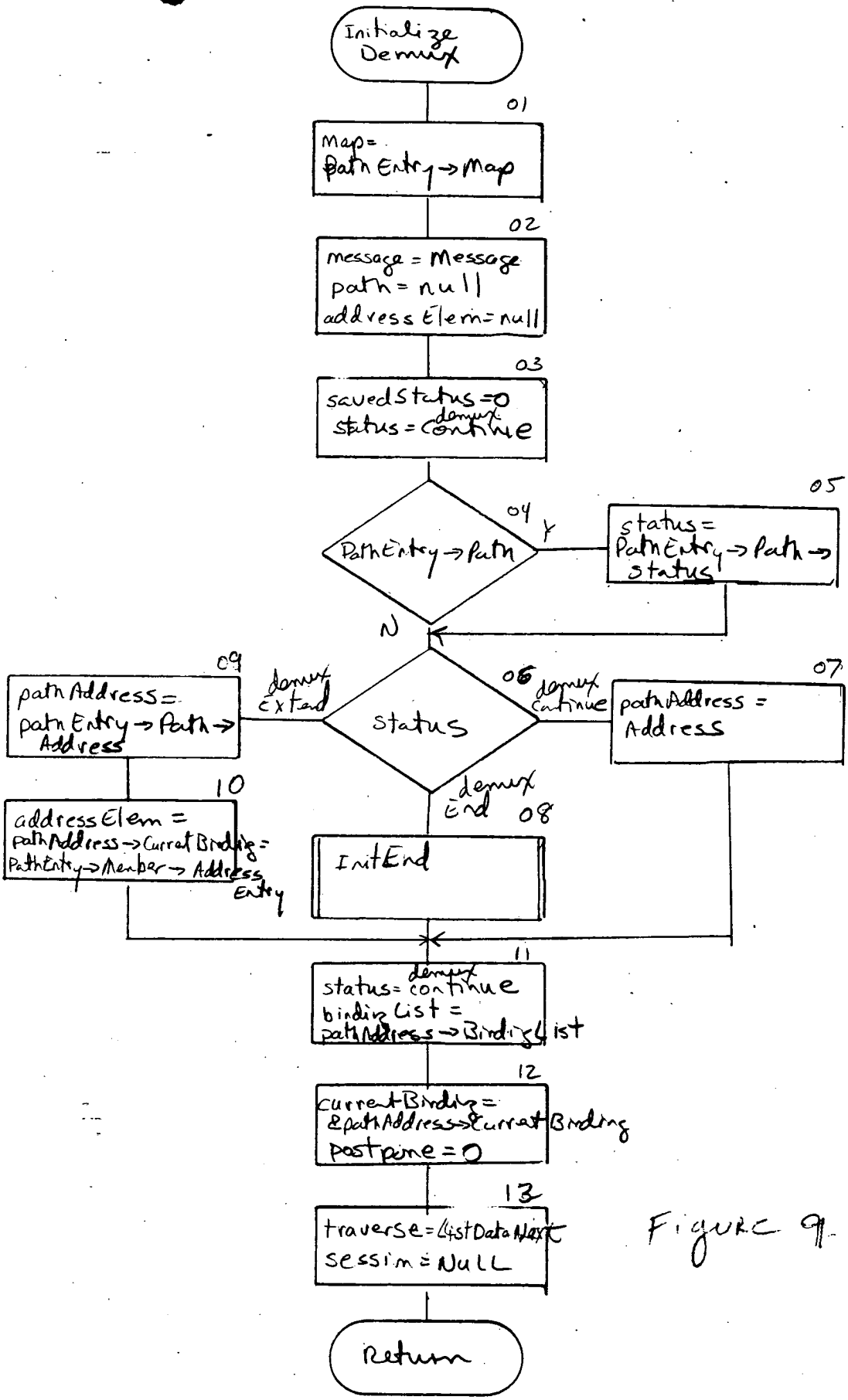

Initialize Demux
01
Map= PathEntry→Map
02
message = Message
path = null
addressElem=null
03
savedStatus=0
status = demux continue
04
PathEntry→Path
Y
05
status = PathEntry→Path→status
N
09
pathAddress = PathEntry→Path→Address
demux Extend
06
Status
demux continue
07
pathAddress = Address
10
addressElem = pathAddress→CurrentBinding = PathEntry→Member→AddressEntry
demux End
08
InitEnd
11
status= demux continue
bindingList = pathAddress→BindingList
12
currentBinding = &pathAddress→currentBinding
postpone = 0
13
traverse=ListData.Next
session=NULL
Return

FIGURE 9

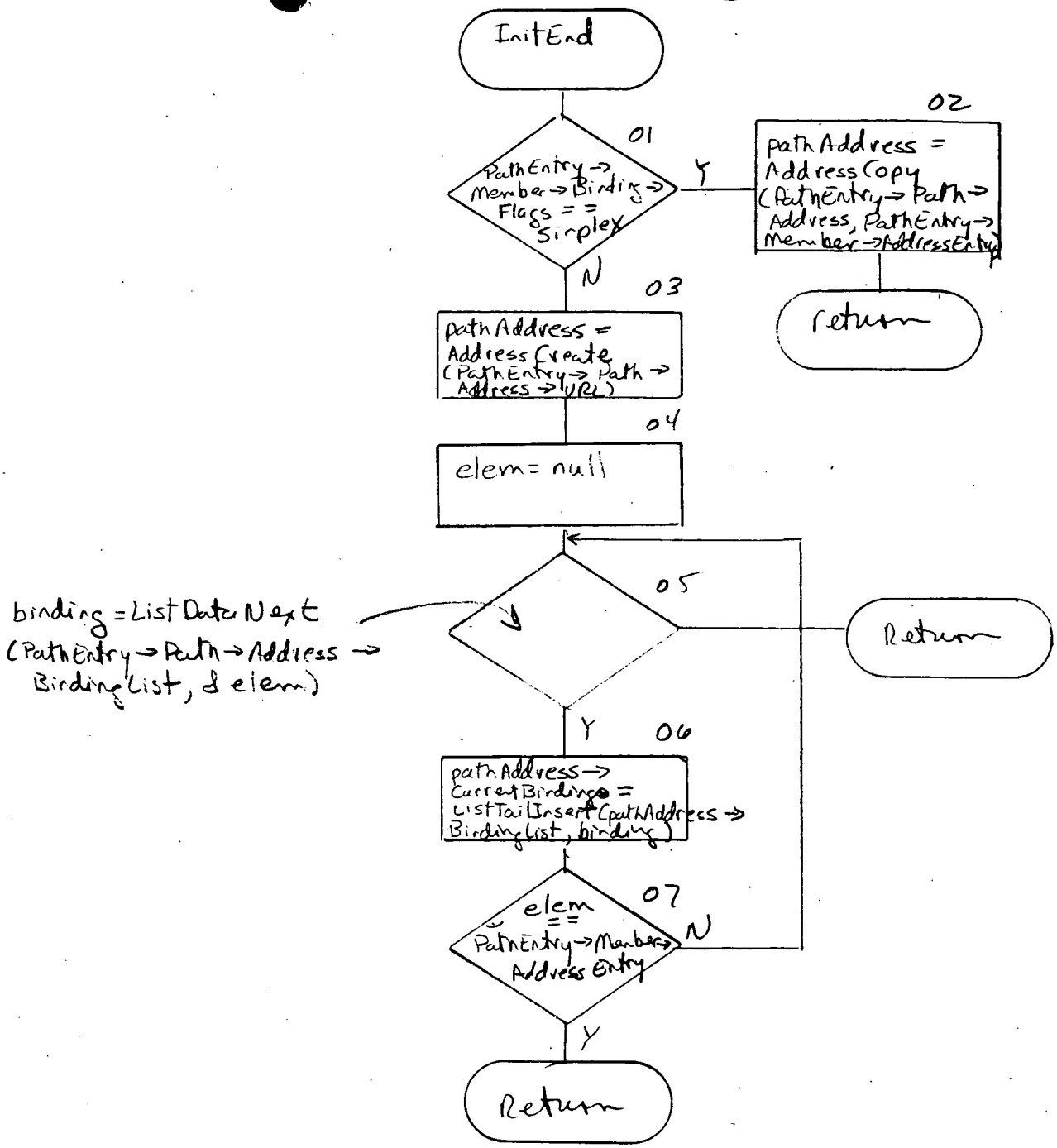

InitEnd
01
PathEntry->Member->Binding->Flags == Simplex
Y
02
pathAddress = AddressCopy(PathEntry->Path->Address, PathEntry->Member->AddressEntry)
return
N
03
pathAddress = AddressCreate(PathEntry->Path->Address->URL)
04
elem = null
05
binding = ListDataNext(PathEntry->Path->Address->BindingList, &elem)
Return
Y
06
pathAddress->CurrentBinding = ListTailInsert(pathAddress->BindingList, binding)
07
elem == PathEntry->Member->AddressEntry
N
Y
Return

Figure 10

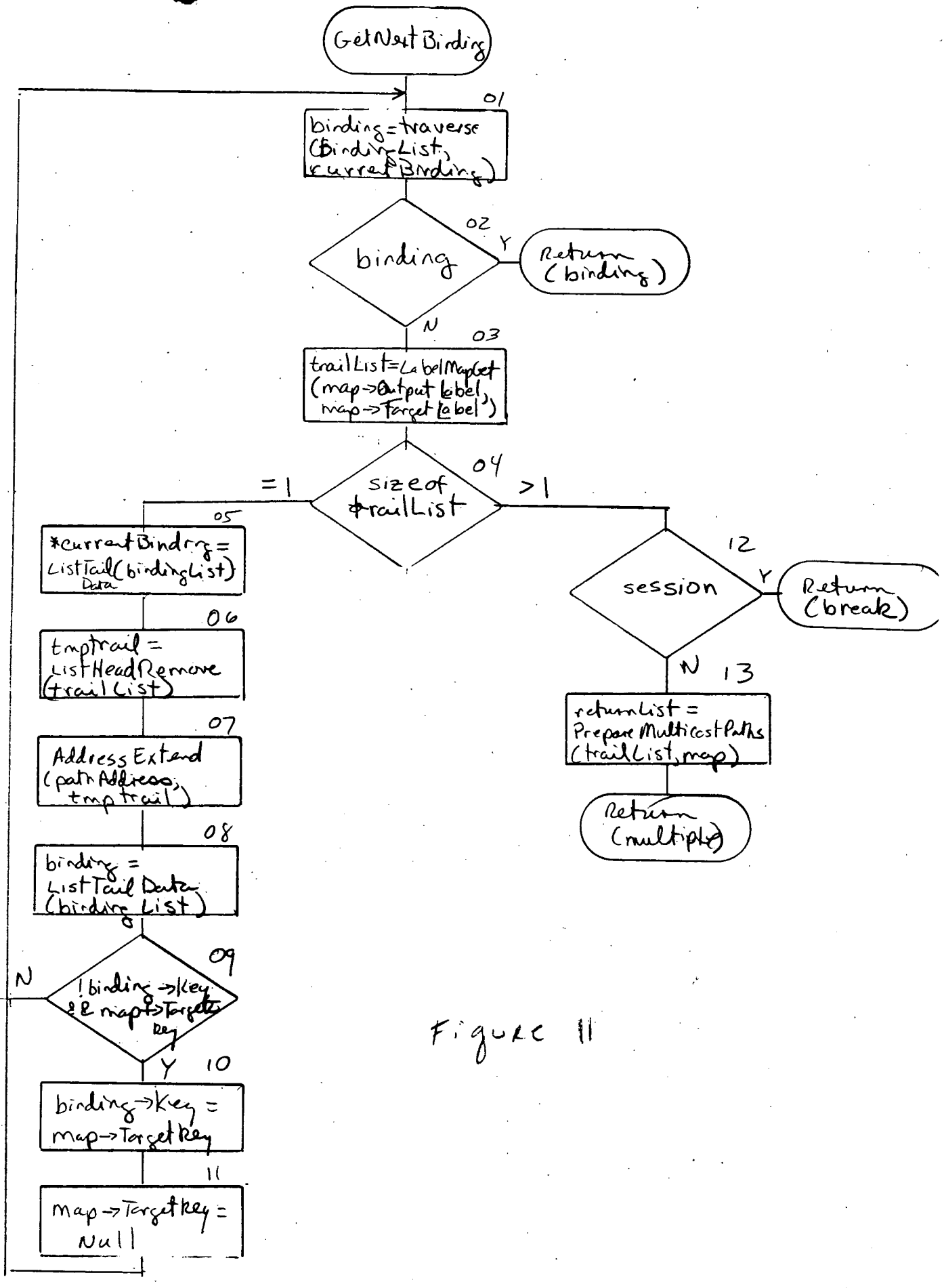

GetNextBinding
01
binding=traverse (BindingList, currentBinding)
02
binding
Y
Return (binding)
N
03
trailList=LabelMapGet (map->Output label, map->Target label)
04
size of *trailList
=1
>1
05
*currentBinding = ListTail(bindingList) Data
06
tmptrail = ListHeadRemove (trailList)
07
Address Extend (pathAddress, tmptrail)
08
binding = ListTail Data (bindingList)
09
N
!binding->key && map->TargetKey
Y
10
binding->key = map->TargetKey
11
map->TargetKey = Null
12
session
Y
Return (break)
N
13
returnList = PrepareMulticastPaths (trailList, map)
Return (multiple)

Figure 11

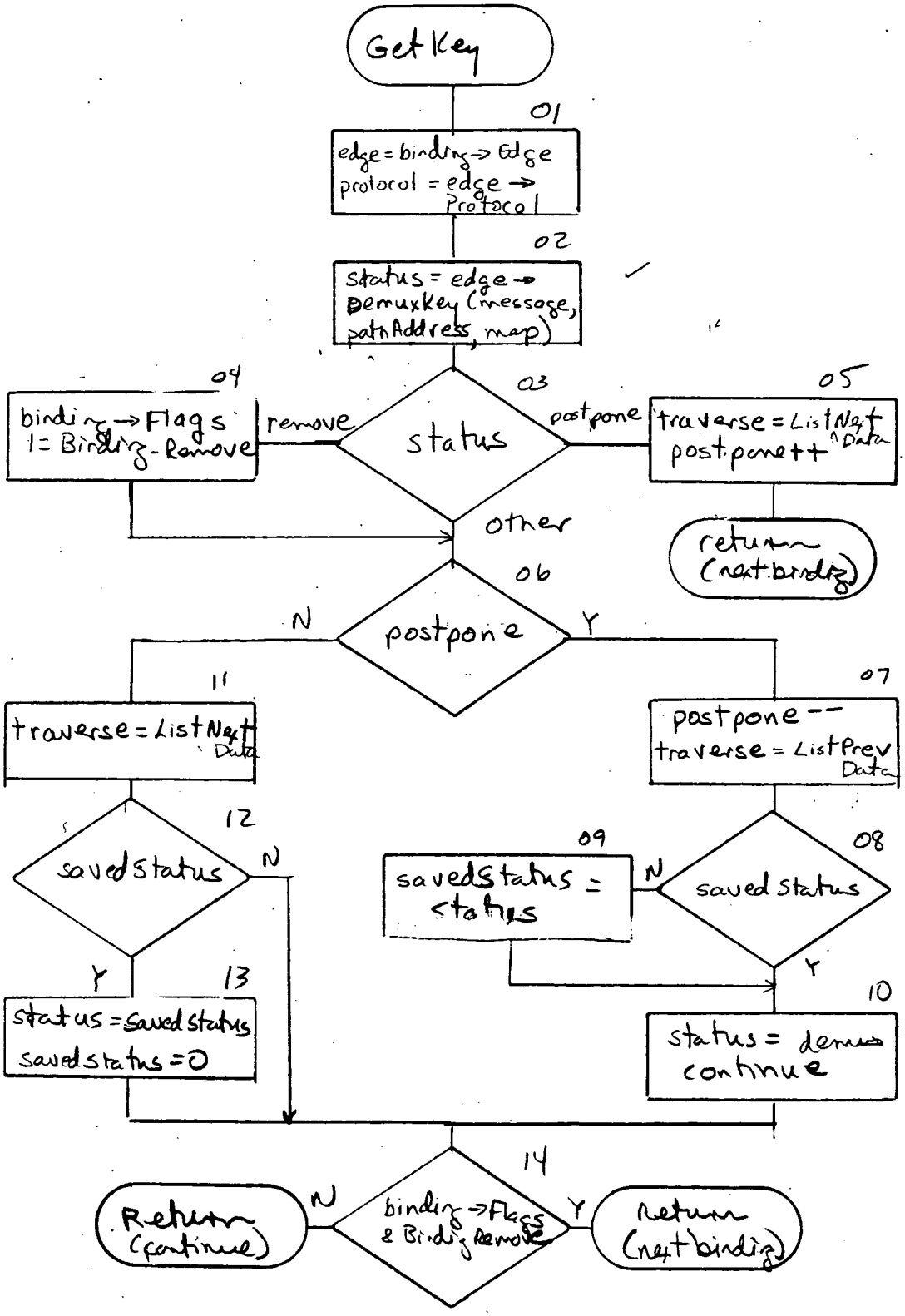

GetKey
01
edge = binding -> Edge
protocol = edge -> Protocol
02
status = edge -> demuxKey (message, pathAddress, map)
03
status
04
binding -> Flags != Binding_Remove
remove
05
traverse = ListNext ^Data
postponeCount++
postpone
other
return (nextBinding)
06
postpone
N
Y
11
traverse = ListNext ^Data
07
postpone --
traverse = ListPrev ^Data
12
savedStatus
N
Y
13
status = savedStatus
savedStatus = 0
09
savedStatus = status
N
08
savedStatus
Y
10
status = demux continue
14
binding -> Flags & Binding_Remove
N
Y
Return (continue)
Return (nextBinding)

Figure 12

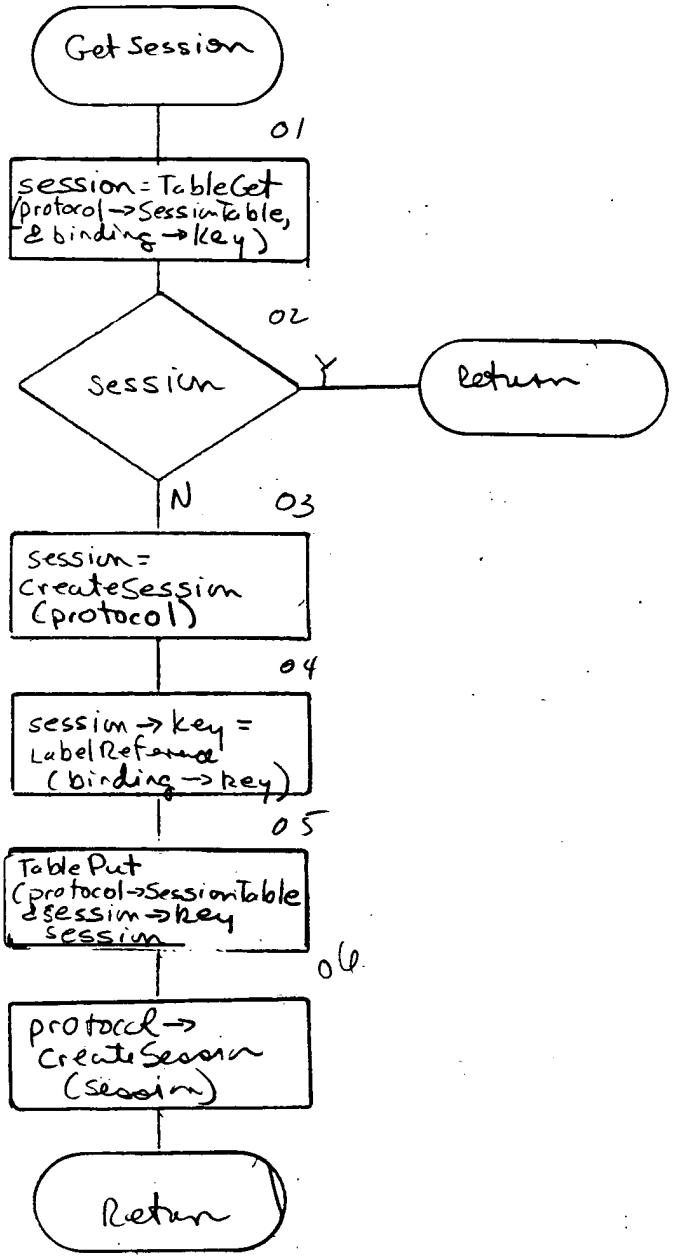

Get Session
01
session = TableGet (protocol->SessionTable, &binding->key)
02
session
Y
Return
N
03
session = CreateSession (protocol)
04
session->key = LabelReference (binding->key)
05
TablePut (protocol->SessionTable &session->key session)
06
protocol-> CreateSession (session)
Return

Figure 13

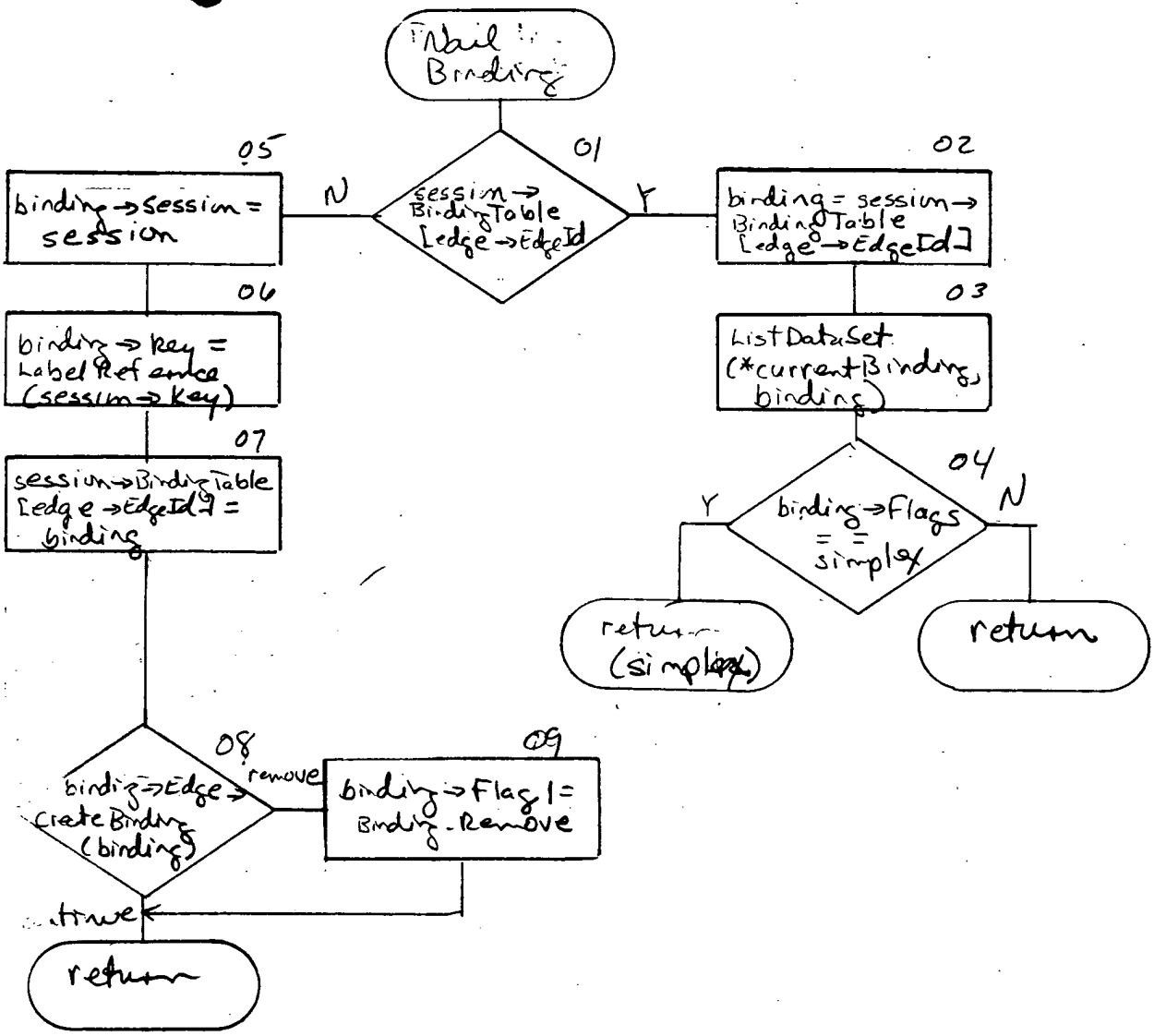

Nail Binding
01
session → BindingTable [edge → EdgeId]
N
Y
05
binding → session = session
02
binding = session → BindingTable [edge → EdgeId]
06
binding → key = LabelReference (session → key)
03
ListDataSet (*currentBinding, binding)
07
session → BindingTable [edge → EdgeId] = binding
04
binding → Flags == simplex
Y
N
return (simplex)
return
08
binding → Edge → CreateBinding (binding)
remove
09
binding → Flag1 = Binding_Remove
true
return

Figure 14

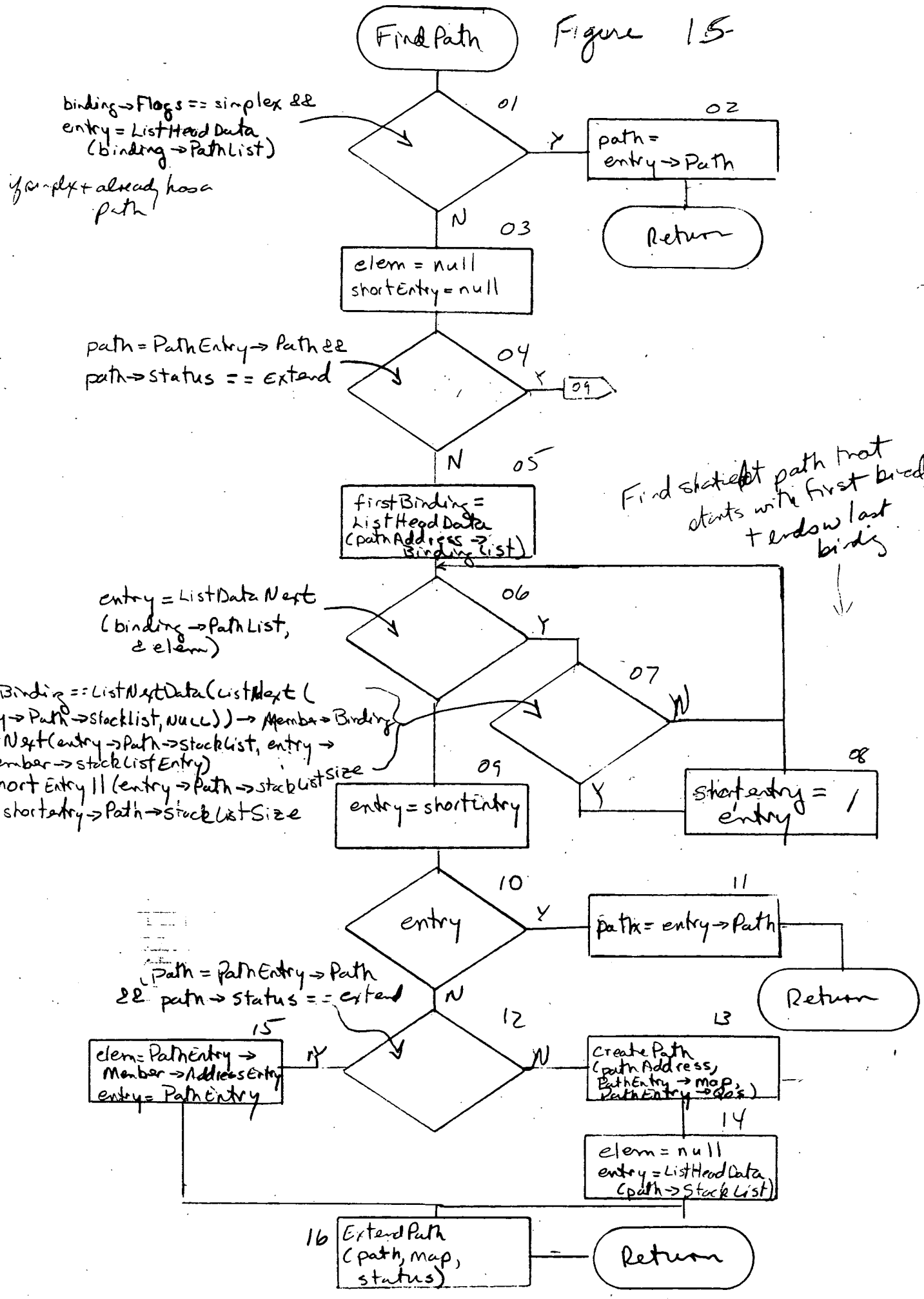


Figure 15
FindPath
01
binding->Flags == simplex &&
entry = ListHeadData
(binding->PathList)
simplex + already has a path
Y
02
path =
entry->Path
Return
N
03
elem = null
shortEntry = null
path = PathEntry->Path &&
path->status == extend
04
Y
09
N
05
firstBinding =
ListHeadData
(pathAddress->
BindingList)
Find shortest path that
starts with first binding
+ endows last
binding
entry = ListDataNext
(binding->PathList,
&elem)
06
Y
07
N
firstBinding == ListNextData(ListNext(
entry->Path->stackList, NULL))->Member->Binding
&& ListNext(entry->Path->stackList, entry->
member->stackListEntry)
&& !shortEntry || (entry->Path->stackListSize
< shortEntry->Path->stackListSize
09
entry = shortEntry
Y
08
shortentry =
entry
10
entry
Y
11
path = entry->Path
Return
N
Path = PathEntry->Path
&& path->status == extend
12
15
elem = PathEntry->
Member->AddressEntry
entry = PathEntry
Y
N
13
CreatePath
(pathAddress,
PathEntry->map,
PathEntry->pa's)
14
elem = null
entry = ListHeadData
(path->StackList)
16
ExtendPath
(path, map,
status)
Return

Path hopping

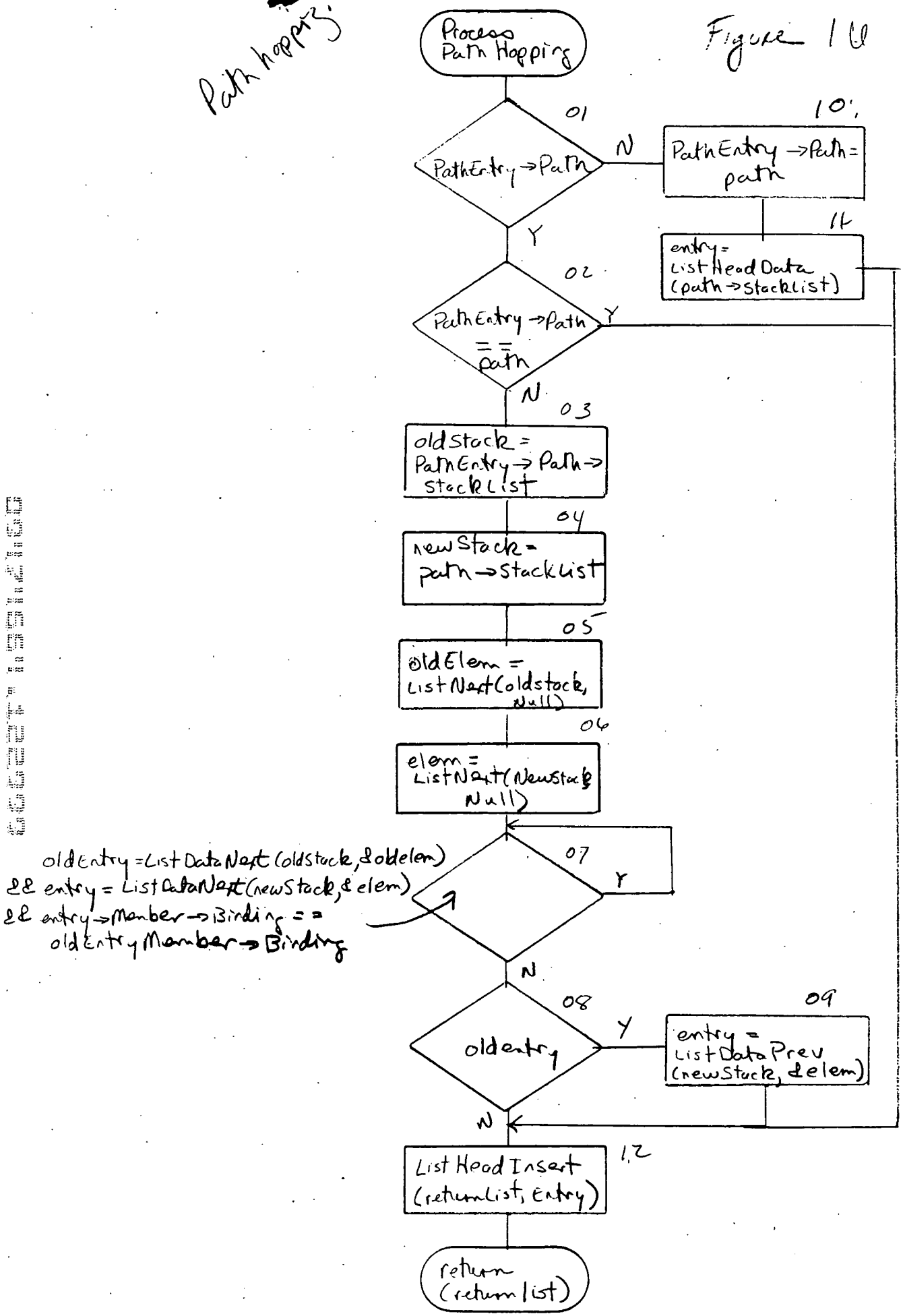



Figure 10